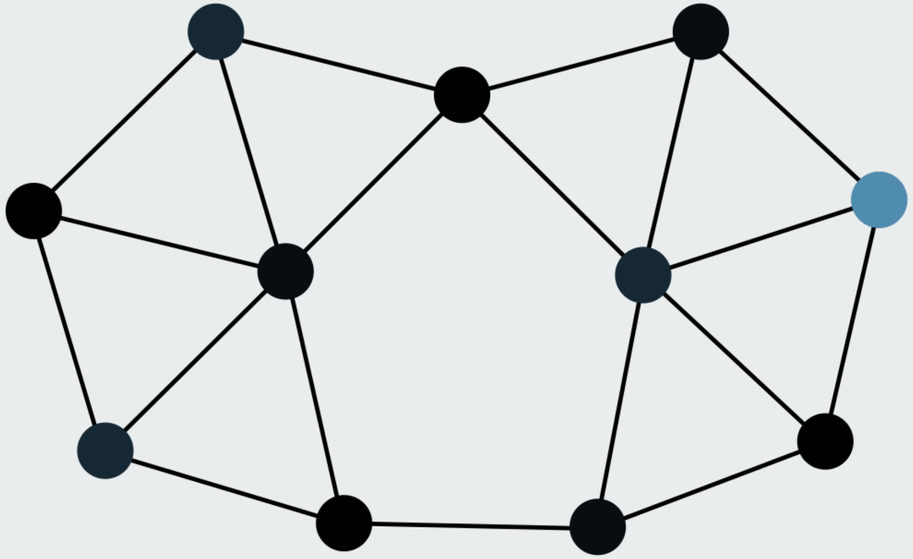
Distributed Systems

Introduction to C++



Jennifer Lebron
jennifer.lebron@griffith.ie

What is C++

General Purpose

- Create all types of programs: games, operating systems, browsers, embedded systems, compilers, all other general purpose software.

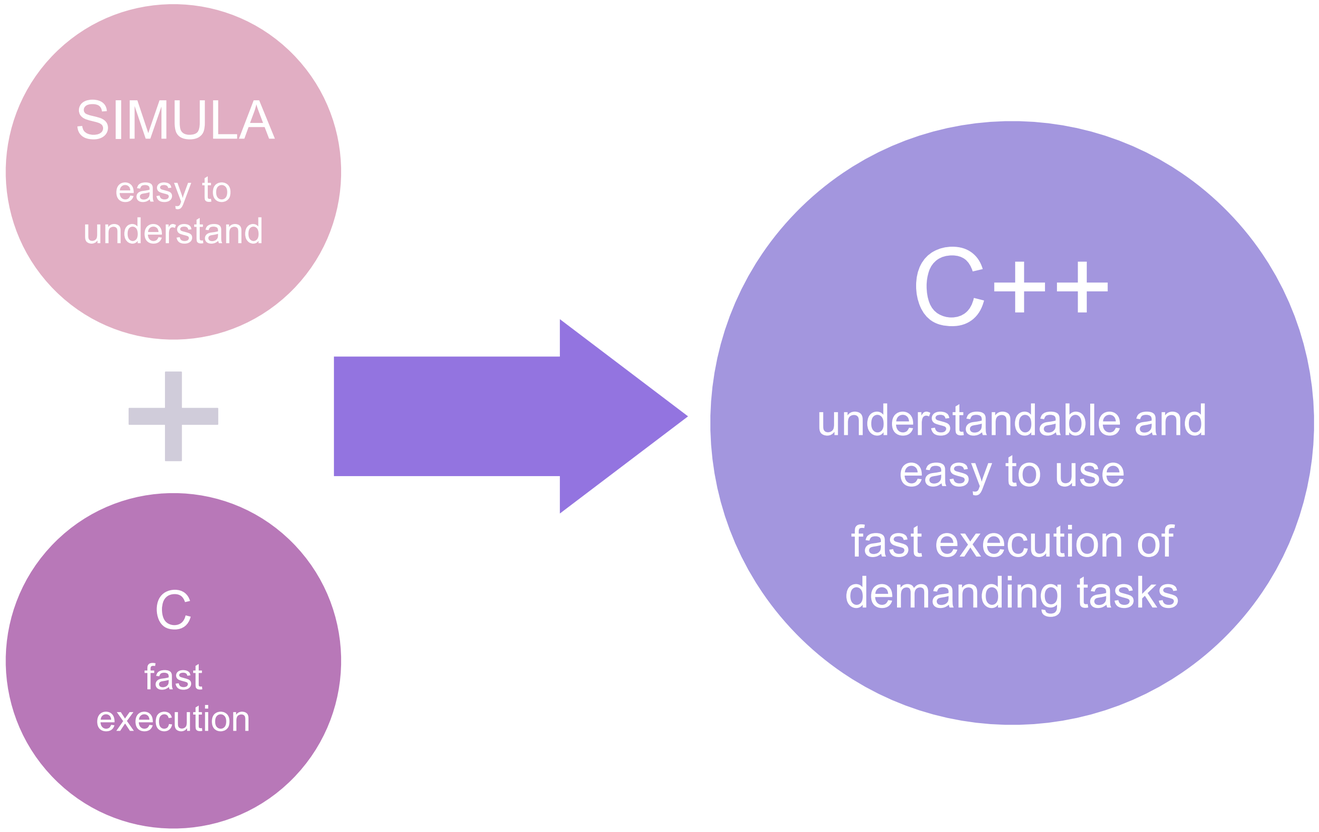
Compiled

- Converting code that you understand into code that the computer understands.
- Building code
- Compile-time errors

Case Sensitive

- C++ distinguishes between uppercase and lowercase letters
- myVariable != myvariable

Why was C++ created?



C++ Overview

C++ Cheatsheet:

https://github.com/mortennobel/cpp-cheatsheet

C++ In depth:

https://www.cplusplus.com/files/tutorial.pdf

Pre-processing

```
                                // Comment to end of line
                                /* Multi-line comment */
#include  <stdio.h>             // Insert standard header file
#include "myfile.h"             // Insert file in current directory
#define X some text             // Replace X with some text
#define F(a,b) a+b              // Replace F(1,2) with 1+2
#define X \
 some text                      // Multiline definition
#undef X                        // Remove definition
```

Literals

Used to express particular values within the source code of a program.

```
255, 0377, 0xff                 // Integers (decimal, octal, hex)
2147483647L, 0x7fffffffl        // Long (32-bit) integers
123.0, 1.23e2                   // double (real) numbers
'a', '\141', '\x61'             // Character (literal, octal, hex)
'\n', '\\', '\'', '\"'          // Newline, backslash, single quote, double quote
"string\n"                      // Array of characters ending with newline and \0
"hello" "world"                 // Concatenated strings
true, false                     // bool constants 1 and 0
nullptr                         // Pointer type with the address of 0
```

Operators

Once we know of the existence of variables and constants, we can begin to operate with them. For that purpose, C++ integrates operators.

Assignment (=)

```
a = 5;  b = a;
```

Arithmetic operators (+, -, *, /, %)

The five arithmetical operations supported by the C++ language

+	addition
-	subtraction
*	multiplication
/	division
%	modulo

Compound Assignment

(+=, -=, *=, /=, %=, >>=, <<=, &=, ^=, |=)

```
value += increase; value = value + increase;
```

(++, --) `c++; c+=1; c=c+1;`

Relational and equality operators (==, !=, >, <, >=, <=)

==	Equal to
!=	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

Logical operators (!, &&, ||)

`NOT, AND, OR`

Conditional operator (?)

returns a value if expression is true and a different one if the expression is false.

Format: `condition ? result1 : result2`

If condition = true, return result1, if not, return result2.

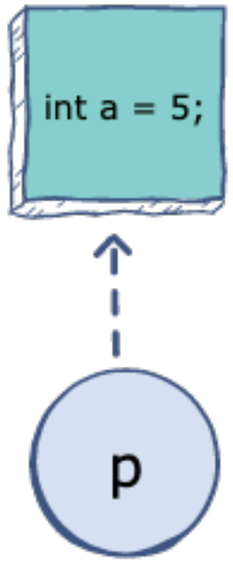
Declarations

```
int x;                          // Declare x to be an integer (value undefined)
int x=255;                      // Declare and initialize x to 255
short s; long l;                // Usually 16 or 32 bit integer (int may be either)
char c='a';                     // Usually 8 bit character
unsigned char u=255;
signed char s=-1;               // char might be either
unsigned long x =
  0xffffffffL;                  // short, int, long are signed
float f; double d;              // Single or double precision real (never unsigned)
bool b=true;                    // true or false, may also use int (1 or 0)
int a, b, c;                    // Multiple declarations
int a[10];                      // Array of 10 ints (a[0] through a[9])
int a[]={0,1,2};                // Initialized array (or a[3]={0,1,2}; )
int a[2][2]={{1,2},{4,5}};      // Array of array of ints
char s[]="hello";               // String (6 elements including '\0')
std::string s = "Hello"         // Creates string object with value "Hello"
std::string s = R"(Hello
World)";                        // Creates string object with value "Hello\nWorld"
int* p;                         // p is a pointer to (address of) int
char* s="hello";                // s points to unnamed array containing "hello"
void* p=nullptr;                // Address of untyped memory (nullptr is 0)
```

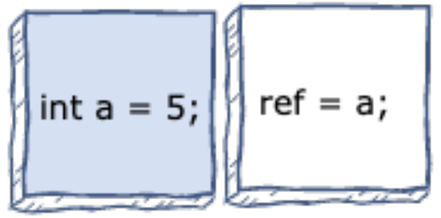
References and Pointers in C++

Pointer - a variable that holds the memory address of another variable.

Reference - alias for an already existing variable. Once a reference is initialised to a variable, it cannot be changed to refer to another variable.



The pointer variable **p** stores the address of the variable **a**; it "points" to the memory location of **a**.



Creating a reference to **a** just makes an alias for it; it does not "point" to **a** by storing its address in a separate memory location

https://www.educative.io/edpresso/differences-between-pointers-and-references-in-cpp

References

A reference must be initialised when it is declared.

References cannot be *NULL*.

References can be used simply, by name.

Once a reference is initialised to a variable, it cannot be changed to refer to a variable object.

& when used with a variable declaration:
"reference to"

```
//    Example:
        int &ra = a;
//    "ra is a reference to a"
```

& when used with an already declared variable:
"address of"

```
  Example:
    &a;
  "address of a"
```

Pointers

Can be initialised to any value anytime after it is declared.

Can be assigned to point to a *NULL* value.

Need to be dereferenced with a *

Can be changed to point to any variable of the same type.

* when used with a variable declaration:
"pointer to"

```
Example:
  int *pa;
"pa is a pointer to an integer"
```

* when used with an already declared pointer:
"dereference"

```
Example:
  std::cout << *pa << std::endl;
"print the underlying value of a"
```

Still unclear? This video explains the concept well: https://www.youtube.com/watch?v=2ybLD6_2gKM

Storage Classes

```
int x;                          // Auto (memory exists only while in scope)
static int x;                   // Global lifetime even if local scope
extern int x;                   // Information only, declared elsewhere
```

Statements

```
x=y;                        // Every expression is a statement
int x;                      // Declarations are statements
;                           // Empty statement
{                           // A block is a single statement
    int x;                  // Scope of x is from declaration to end of block
}
if (x) a;                   // If x is true (not 0), evaluate a
else if (y) b;              // If not x and y (optional, may be repeated)
else c;                     // If not x and not y (optional)

while (x) a;                // Repeat 0 or more times while x is true

for (x; y; z) a;            // Equivalent to: x; while(y) {a; z;}

for (x : y) a;              // Range-based for loop e.g.
                            // for (auto& x in someList) x.y();


do a; while (x);            // Equivalent to: a; while(x) a;
```

Functions

```
int f(int x, int y);          // f is a function taking 2 ints and returning int
void f();                     // f is a procedure taking no arguments
void f(int a=0);              // f() is equivalent to f(0)
f();                          // Default return type is int
inline f();                   // Optimize for speed
f() { statements; }           // Function definition (must be global)
T operator+(T x, T y);        // a+b (if type T) calls operator+(a, b)
T operator-(T x);             // -a calls function operator-(a)
T operator++(int);            // postfix ++ or -- (parameter ignored)
extern "C" {void f();}        // f() was compiled in C
```

Functions

Function parameters and return values may be of any type. A function must either be declared or defined before it is used. It may be declared first and defined later.

Every program consists of a set of a set of global variable declarations and a set of function definitions (possibly in separate files), one of which must be:

```
int main()  { statements... }     // or
int main(int argc, char* argv[]) { statements... }
```

`argv` is an array of `argc` strings from the command line. By convention, `main` returns status `0` if successful, 1 or higher for errors.

Functions with different parameters may have the same name (overloading).

Input/Output

C++ uses a convenient abstraction called streams to perform input and output operations in sequential media such as the screen or the keyboard. A stream is an object where a program can either insert or extract characters to/from it.

The standard C++ library includes the header file `iostream`, where the standard input and output stream objects are declared.

Output

By default, the standard output of a program is the screen, and the C++ stream object defined to access it is **cout**.

`cout` is used in conjunction with the **insertion operator**, which is written as <<

```
cout << "Output sentence"; // prints Output sentence on screen
cout << 120;               // prints number 120 on screen
cout << x;                 // prints the content of x on screen
```

The insertion operator (<<) may be used more than once in a single statement:

```
cout << "Hello, " << "I am " << "a C++ statement";
```

To add a new line, you can use the `endl` manipulator. For example:

```
cout << "First sentence." << endl;
cout << "Second sentence." << endl;
```

Would output:

```
First sentence.
Second sentence.
```

Output

`cout` is a reserved name used by multiple different libraries. In order to use it you will need to inform the compiler which library you are referencing cout from. In our case ‘std’ from the `<iostream>` library.

This can be done in one of two ways:

Reference ‘std’ inline (recommended):

You can do this inline by including `std::` before the `cout` and `endl` commands.

```
std::cout << "Hello world!";
std::cout << "Amount of nodes: " << x <<
std::endl;
std::cout << "Goodbye!" << std::endl;
```

Set the namespace:

To set the namespace to std, add this line underneath your ‘include’ imports:

```
using namespace std;
```

You will then be able to use cout and endl without adding ‘std::’ beforehand.

Warning:
Setting the namespace to std can cause issues with reserved names.

Input

The standard input device is usually the keyboard. Handling the standard input in C++ is done by applying the overloaded operator of extraction (>>) on the **cin** stream.

The operator must be followed by the variable that will store the data that is going to be extracted from the stream. For example:

```
int age;
cin >> age;
```

// declares a variable of type int called age
// waits for an input from cin (the keyboard) to store it in this integer variable.

`cin` can only process the input from the keyboard once the `RETURN` key has been pressed.

Always consider the type of the variable that you are using as a container.

You can also use cin to request more than one datum input from the user:

```
cin >> a >> b;
```

is equivalent to

```
cin >> a;
cin >> b;
```

The user must give two data, one for variable *a* and another one for variable *b* that may be separated by any valid blank separator: a space, a tab character or a newline.